



COV-1

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE
BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Applicants/Appellants : Benjamin E. Chelf et al.
Application No. : 10/637,453
Confirmation No. : 3626
Filed : August 8, 2003.
For : SYSTEMS AND METHODS FOR PERFORMING
STATIC ANALYSIS ON SOURCE CODE

Group Art Unit : 2192
Examiner : Eric B. Kiss

Mail Stop Appeal Brief - Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

APPEAL BRIEF

Sir:

Applicants are filing this brief pursuant to 37 C.F.R.
§ 41.37 in support of their appeal from the rejection of claims
1-18 in the Office Action dated May 31, 2007. A Notice of
Appeal for this case is being filed concurrently herewith.

07/17/2007 MGE BREM1 00000104 10637453

02 FC:2402

250.00 OP

Introduction

Claims 1-18 were rejected in an Office Action dated
May 31, 2007. Claims 1-10 and 15-18 were said to be anticipated

by the Smatch C source checker as evidenced by various Smatch documents. Claims 11-14 were said to be unpatentable over Granston et al. U.S. Patent No. 5,960,202 and the Smatch documents. For the reasons set forth below, the Board should find these rejections to be in error and should reverse the Examiner. A copy of claims 1-18 is attached as an Appendix.

Real Party In Interest

The real party in interest is Coverity, Inc.

Related Appeals and Interferences

There are no other appeals or interferences known to applicants or their legal representative that will directly affect or be directly affected by or have a bearing on the Board's decision in the pending appeal.

Status of Claims

Claims 1-18 have been rejected in this application and are on appeal.

Status of Amendments

No amendments have been filed subsequent to the May 31, 2007 final Office Action in this case.

Summary of the Claimed Subject Matter

Applicants' invention relates to ways to perform static error analysis in complex software environments.

As applicants explain in their specification, static error analysis tools operate on code that is not running during the analysis process. Although static error analysis tools make certain simplifying assumptions, static analysis tools can detect errors that are often missed when using dynamic analysis tools alone. As an example, static analysis may detect an illegal operation that is contained in a rarely traversed or otherwise hard-to-test conditional branch path.

Although it is desirable to use static error analysis tools, there are obstacles to performing static error analysis in modern software development environments. For example, in many modern software development environments, source code, compilers, and ancillary software components are distributed across various different directories and systems.

As a result of this complexity, software developers typically use build management utilities such as the "Make" program to assist in the process of building executable code. Software developers also construct their own build management utilities (e.g., by writing customized shell scripts). As the software development environment at a given organization

matures, it is commonplace for the overall software build process used by that organization to involve many directory changes, variable assignments, calls of different build management utilities, etc.

The use of static analysis debugging tools in this type of complex build environment is difficult. Static analysis software tools operate on source code, so all of the relevant source code to be debugged must be identified. Different static analysis tools may be used in place of different compilers, so it may be necessary to identify which compilers are normally used to compile which source code files during the build process. Although it may be possible for the proper static analysis tools to be selected and invoked manually, this process can be time consuming and prone to errors, particularly when it is necessary to set up and run a new suite of static analysis tools in a relatively short period of time in a complex build environment.

To address these problems with the conventional deployment of static error analysis tools, applicants have developed improved techniques for using static analysis tools to perform static error analysis on source code files.

There are four independent claims in applicant's patent application (claims 1, 7, 11, and 15), each of which is directed to a different technique for using static analysis

tools to perform static error analysis.

Claims 1-6

A first aspect of the invention is defined by independent claim 1. With the arrangement of independent claim 1, a build program is run on a computer system to invoke compilers on the computer system that compile source code files into executable code. The process of running the build program produces a build program output. As specified by claim 1, a static analysis tool management program is run on the computer system to invoke static analysis tools. This produces corresponding static error analysis results. Claim 1 requires that the static analysis tool management program accept the source code and the build program output as inputs.

As shown in FIG. 1 of applicants' specification, build program 12 invokes compilers 20 on computer system 10 that compile source code 14. Static analysis management program 22 is run on computer system 10 to invoke static analysis tools 24. As indicated by line 76 on the left of FIG. 1, build program 12 produces a build program output that is received by static analysis tool management program 22 as an input. As indicated by line 76 on the right of FIG. 1, static analysis tool management program 22 also accepts source code 14 as an input.

Claims 7-10

A second aspect of the invention is defined by independent claim 7. Claim 7 is directed toward a method for using static analysis tools on a computer system to perform error analysis on source code files. The source code files may be compiled on the computer system using compilers having corresponding compiler names that are invoked using a build program during a build process. The method of claim 7 involves creating a new directory on the computer system. The method of claim 7 also involves modifying the search path on the computer system so that the new directory is included first. The static analysis tools are placed into the new directory and are given names matching the compiler names. Static analysis is performed by running the build program so that the static analysis tools with the names matching the compiler names are invoked.

The method of claim 7 is shown in FIG. 6 of applicants' patent application and is described at page 24, line 1 to page 28, line 27.

The step of creating the new directory on the computer system is illustrated as step 86 in FIG. 6.

The step of modifying the search path on the computer system so that the new directory is included first in the search path is illustrated as step 88.

The operations involved in placing the static analysis tools into the new directory and in giving the static analysis tools in the new directory names matching the compiler names are illustrated as steps 90 and 92.

The step of running the build program so that the static analysis tools with the names matching the compiler names are invoked is illustrated as step 94.

With the method of claim 7, it is not necessary to use a static analysis tool management program 22 to invoke the static analysis tools 24, because the changes to the computer system's path and the changes to the names of the static analysis tools are sufficient to properly invoke the necessary static analysis processes.

Claims 11-14

A third aspect of the invention is defined by independent claim 11. Claim 11 is directed toward a method for using static analysis tools on a computer system to perform error analysis. With the method of claim 11, error analysis is performed on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process. The computer system has an operating system. The method of claim 11 involves running the build program on the computer system. The method of claim 11

also involves running a monitoring program on the computer system while the build program is running to compile the source code files. The monitoring program monitors activity between the build program and the operating system. The output from the monitoring program is used to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files.

The steps involved in using this technique to perform static analysis are shown in FIG. 7 of applicants' patent application. As shown in FIG. 7, the monitoring program (also sometimes referred to as a trace program) is started at step 98. At step 100, the build program is run while using the trace program to monitor activity between the build program and the operating system. The information captured by the trace program is used to run a version of the build program with static analysis tools substituted for compilers (step 104).

Claims 15-20

A fourth aspect of the invention is defined by independent claim 15. Claim 15 is directed toward a method for using static analysis tools on a computer system to perform error analysis. The error analysis is performed on source code files that may be compiled on the computer system using

compilers that are invoked using a build program during a build process. The method of claim 15 involves redefining operating system commands in the operating system. The build program is run on the computer system. As defined by claim 15, the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers so that the error analysis on the source code files is performed.

The method of claim 15 is shown in FIG. 8 and is described at page 30, line 30 to page 32, line 18.

The step of redefining the operating system commands in the operating system is illustrated as step 108 of FIG. 8.

The step of running the build program on the computer system so that the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers and so that the error analysis on the source code files is performed is illustrated as step 110 of FIG. 8.

Grounds of Rejection to be Reviewed on Appeal

Claims 1-10 and 15-18 were rejected under 35 U.S.C. §102(a) as being anticipated by the Smatch C source checker ("Smatch") as evidenced by the documents Installing Smatch!!! ("SmInst"), Smatch!!! ("SmMain"), Smatch Intermediate Code Representation!!! ("SmIR"), Using Smatch!!! ("SmUsing"), and Using Smatch.pm!!! ("SmUpm"). Claims 11-14 were rejected under

35 U.S.C. §103(a) as being unpatentable over U.S. Patent No. 5,960,202 (Granston et al.) and the Smatch documents.

Applicants are appealing these rejections of claims 1-18.

In particular, applicants are appealing each of the following rejections:

1. The rejection of independent claim 1 and its dependent claims 2-6 under 35 U.S.C. §102(a).

2. The rejection of independent claim 7 and its dependent claims 8-10 under 35 U.S.C. §102(a).

3. The rejection of independent claim 11 and its dependent claims 12-14 under 35 U.S.C. §103(a).

4. The rejection of independent claim 15 and its dependent claims 16-18 under 35 U.S.C. §102(a).

Argument

In the May 31, 2007 Office Action, claims 1-18 were rejected under 35 U.S.C. § 102(a) and 103(a) as set forth above. These rejections are improper and should be reversed.

Each of applicants' four independent claims is directed to a different aspect of applicants' invention. Applicants are therefore presenting separate arguments for each of the four independent claims and their corresponding dependent

claims. Each of these four arguments is presented under its own heading.

Argument for Independent Claim 1 and its Dependent Claims 2-6

Claims 1-6 were rejected under 35 U.S.C. §102(a) as being anticipated by the Smatch documents. These rejections are improper and should be reversed.

With the arrangement of independent claim 1, a build program is run on a computer system to invoke compilers on the computer system that compile source code files into executable code. The process of running the build program produces a build program output. As shown in FIG. 1 of applicants' specification, build program 12 and compilers 20 are separate from each other. Because language of claim 1 explicitly requires that the build program invoke "compilers," claim 1 is directed to situations in which a build program invokes more than one compiler.

As specified by claim 1, a static analysis tool management program is run on the computer system to invoke static analysis tools. This produces corresponding static error analysis results. The static analysis tool management program 22 is shown in FIG. 1. Claim 1 requires that the static analysis tool management program accept source code and the build program output as inputs. Lines 76 in FIG. 1 illustrate

how the output of build program 12 and source code 14 are provided to static analysis tool management program 22 as inputs.

Nothing like this arrangement is shown or suggested by the Smatch documentation.

As described on page 1 of SmMain, Smatch uses a modified gcc compiler to generate .c.sm files. The .c.sm files are analyzed by Smatch scripts that print out error messages. The modified gcc compiler accepts source code as an input and produces .c.sm files (and executable code) as preprocessed output. Smatch's checker scripts process the preprocessed output from the modified gcc compiler to produce static analysis results.

In the Office Action, the Examiner suggests that Smatch's modified gcc compiler is that same as a build program. This is incorrect. Build programs such as the Make program are programs that invoke appropriate compilers to compile source code files. The source code files may be in different languages, so the build program often invokes multiple compilers suitable for compiling source code files in multiple languages. Build programs may also call a number of build management utility programs. Build programs do not perform the functions of a compiler nor do compilers perform the functions of build programs. Build programs and compilers are simply two different

types of programs. The distinction between these two types of programs are well known and are emphasized in applicant's specification. For example, FIG. 1 depicts build program 12 and compilers 20 as being separate. Moreover, the cited prior art (e.g., Granston) describes arrangements in which build programs are separate from compilers and are used to invoke compilers, consistent with normal usage of the terms "build program" and "compilers."

Because build programs are programs that invoke compilers rather than being compilers themselves, it is not correct to equate Smatch's modified gcc compiler with applicants' claimed build program as urged in the Office Action. Claim 1 also explicitly requires that the build program invoke multiple compilers on the computer system that compile source code files into executable code. Because the modified gcc compiler in Smatch does not invoke a single compiler, let alone invoke "compilers" as required by claim 1, the modified gcc compiler in Smatch does not meet the requirements of claim 1.

Smatch also fails to show or suggest "running a static analysis tool management program to invoke the static analysis tools and produce corresponding static error analysis results" as required by claim 1. In the Office Action, it was suggested that one of the Smatch checker scripts could be considered to be the same as applicants' claimed static analysis tool management

program. Applicants disagree.

As described in SmMain, the .c.sm files are "piped through individual Smatch scripts." These scripts are not called by a static analysis tool management program. Rather, an individual types in a command to invoke a desired script. This is set forth in the first paragraph of SmIR, which explains how one types `"/checker_script.pl file.c.sm"` to invoke a desired checker script. The checker scripts in Smatch are therefore manually invoked as needed by a user, rather than being invoked by a static analysis tool management program. Moreover, each checker script is not a separate static analysis tool management program, because the checker scripts process the .sm files, they do not "invoke static analysis tools" as required by claim 1.

Claim 1 is therefore directed toward a method that involves running a build program that invokes compilers, whereas the Smatch modified gcc compiler is not a build program and does not invoke compilers. Claim 1 is also directed toward a method that runs a static analysis tool management program to invoke static analysis tools, whereas Smatch's manually invoked checker scripts are not static analysis tool management programs and do not invoke static analysis tools. For at least these reasons, the Smatch documents do not disclose the features of claim 1. Claim 1 is therefore not anticipated by the Smatch documents and is in condition for allowance. Claims 2-6 depend from claim 1

and are allowable because claim 1 is allowable.

Applicants presented these arguments to the Patent Office in their March 18, 2007 Reply to Office Action. In the May 31, 2007 Office Action, the Patent Office responded. According to the Examiner's response, the term "build program" in claim 1 has been interpreted to mean "any program that assists in the process of building executable code." The Examiner states that the modified compiler of Smatch, xgcc, is therefore a build program. The Examiner further asserts that running the xgcc program of Smatch invokes a compiler (a modified gcc compiler) to compile code and produce .sm files.

According to the Examiner, the xgcc program in the Smatch documents is a build program and the Smatch scripts are each a static analysis tool management program. This argument is unconvincing.

Assume, for the sake of argument, that Smatch's xgcc program is a "build" program and that Smatch's scripts are each a static analysis tool management program. Even making these assumptions, the Smatch documents fail to show or suggest the arrangement defined by claim 1. In particular, the xgcc program does not invoke any compilers. This is because the xgcc program is a compiler. There is therefore no need for the xgcc program to invoke a separate compiler program to perform its functions.

Claim 1 requires a build program that invokes

compilers. As best as applicants' can determine from the Office Action, the Examiner is asserting that the xgcc program meets this limitation by invoking itself. However, a program does not invoke itself. When a program is run, it may either perform functions by itself or may invoke other programs to perform certain functions on its behalf. Because the xgcc compiler in Smatch does not invoke a compiler as required by claim 1, the xgcc compiler does not perform the function of the build program in claim 1, whether or not the xgcc compiler is considered to be a build program.

Moreover, applicants have made it clear that the method of claim 1 involves using a build program to invoke multiple compilers. The language of claim 1 states explicitly that the method of claim 1 involves "running a build program on the computer system to invoke compilers." By using the plural term "compilers," applicants have precluded the type of misreading of the claim language that is being proposed by the Examiner. According to the language of claim 1, the build program must invoke more than one compiler. Applicants explained this claim feature to the Examiner in their March 16, 2007 Reply. The Examiner has not even attempted to rebut this point in the May 31, 2007 Office Action.

Another reason that claim 1 is allowable relates to the step of "running a static analysis tool management program

on the computer system to invoke the static analysis tools." In the May 31, 2007 Office Action, the Examiner has taken the position that an individual checker script in Smatch is the same as applicants' claimed static analysis tool management program. However, if the Smatch checker script were to be a static analysis tool management program as asserted by the Examiner, that Smatch checker script would still fail to satisfy the explicit language of claim 1. In particular, the Smatch checker script would fail to invoke even one of the required static analysis tools. The Smatch checker scripts simply do not invoke any other tools, let alone is a single Smatch checker scrip used to invoke multiple tools as required when running the static analysis tool management program defined by claim 1.

In the May 31, 2007 Office Action, the Examiner suggests that a Smatch checker script could fulfill the claimed limitation of "invoking static analysis tools," by saying that there is functionality coded into the Smatch checker script and by suggesting that this functionality is the same as the static analysis tools. With this argument, the Examiner has again ignored the explicit language of claim 1. Claim 1 requires the step of "running a static analysis tool management program on the computer system to invoke the static analysis tools." The Smatch scripts do not invoke any tools. The script functionality that is coded into the scripts is described in

Smatch documents such as SmMain. As described in SmMain, a Smatch script might be coded to include functionality to look for code that makes a call `copy_to_user()` while the kernel is locked. This functionality is built into the code of a Smatch script. Accordingly, a given Smatch script does not need to invoke a static analysis tool, let alone does a given Smatch script need to invoke multiple static analysis tools, as required by claim 1.

The Smatch documents simply do not disclose use of a build program that invokes compilers or a static analysis tool management program that invokes static analysis tools. Through unconvincing logic, the Examiner has attempted to equate the Smatch compiler `xgcc` with applicants' claimed build program. But doing so fails, because `xgcc` does not invoke a compiler, let alone does `xgcc` invoke compilers. Similarly, although the Smatch documents fail to disclose a static analysis tool management program, the Examiner has attempted to equate a Smatch checker script with applicants' claimed static analysis tool management program. The problem with this argument is that running the Smatch checker script does not invoke static error analysis tools as required by claim 1.

For each of these reasons, claim 1 is allowable, as are claims 2-6, which depend from claim 1. The rejections of claims 1-6 are therefore improper and should be reversed.

Argument for Independent Claim 7 and its Dependent Claims 8-10

Claims 7-10 were rejected under 35 U.S.C. §102(a) as being anticipated by the Smatch documents. These rejections are improper and should be reversed.

Claim 7 is directed toward a method for using static analysis tools on a computer system to perform error analysis on source code files. The source code files may be compiled on the computer system using compilers having corresponding compiler names that are invoked using a build program during a build process. The method of claim 7 involves creating a new directory on the computer system. The method of claim 7 also involves modifying the search path on the computer system so that the new directory is included first. The static analysis tools are placed into the new directory and are given names matching the compiler names. Static analysis is performed by running the build program so that the static analysis tools with the names matching the compiler names are invoked.

Smatch uses a different approach. In the Smatch arrangement, the original compiler is replaced by a modified compiler. The modified compiler is invoked in place of the original compiler. To ensure that the modified compiler is invoked properly, the modified compiler is given a different name than the original compiler. The name of the original

compiler is gcc. The name of the modified compiler is xgcc. In the user's Makefile, the user comments out the original variable cc using the comment symbol "#" and inserts a new variable that refers to the modified compiler name xgcc. This process is described in step 1a) of SmUsing.

With the Smatch approach, there is no need to modify the search path as required by claim 7, because the modified compiler is called using the variable cc. Smatch therefore does not show or suggest modifying a search path on a computer system so that a new directory is included first in a search path as required by claim 7.

Moreover, with the Smatch approach, the modified compiler that is called has a different name than the original compiler. The name of the modified compiler is xgcc, whereas the name of the original compiler is gcc. With the method of claim 7, the static analysis tools are placed into a new directory and are given names matching the compiler names. Static analysis is then performed by running a build program so that the static analysis tools with the names matching the compiler names are invoked. Unlike the Smatch approach, applicants' claimed method uses static analysis tools with names that match the compiler names. In the Office Action, it was suggested that the modified compiler xgcc in Smatch is a static analysis tool having a name that matches a compiler name. This

is incorrect, since the name xgcc is different than the compiler name gcc.

Because claim 7 requires modification of a search path so that a new directory is included first in a search path, whereas Smatch makes no search path modifications, claim 7 is not anticipated by Smatch. Claim 7 is also not anticipated by Smatch because claim 7 requires static analysis tools having names that match compiler names, whereas no such matching names are present in Smatch.

In the May 31, 2007 Office Action, the Examiner has asserted that the variable cc in Smatch represents a name for the xgcc compiler (which is presumably being equated with a static analysis tool) and the gcc compiler. This argument is not persuasive. Variable cc is not a compiler name. The name of the compiler that is disclosed in Smatch is gcc.

Moreover, claim 7 explicitly requires that a search path on the computer system be modified so that a newly created directory is included first in the search path. By invoking gcc and xgcc through the unmodified/modified versions of variable cc, the arrangement of Smatch avoids the need to make any modifications to the computer system search path. Invoking programs from different locations on a computer as needed is simply not the same as the claimed step of "modifying a search path on the computer system so that the new directory is

included first in the search path." When a search path is modified, subsequent searches for a program follow the modified search path. When using the variable cc, in contrast, a program is directly invoked from its directory, without any need to modify the computer system's search path. The arrangement of Smatch therefore obviates the need to change the computer system's search path and thus teaches away from using the approach defined by claim 7.

Claim 7 also explicitly requires that multiple static analysis tools be placed in the new directory. In Smatch, there is only a single modified compiler xgcc. Because there is only a single modified compiler xgcc in Smatch, Smatch does not disclose "placing the static analysis tools in to the new directory" as required by claim 7.

Claim 7 includes a step of "running the build program so that the static analysis tools with the names matching the compiler names are invoked." In Smatch, there is only a single xgcc compiler, so that multiple static analysis tools are not invoked, let alone multiple static analysis tools with names matching compiler names.

For each of these reasons, claim 7 is not anticipated by Smatch and is allowable. Claims 8-10 depend from claim 7 and are allowable for the same reasons that claim 7 is allowable. The Examiner's rejections of claims 7-10 are in error and should

be reversed.

Argument for Independent Claim 11 and its Dependent Claims 12-14

Claims 11-14 were rejected under 35 U.S.C. §103(a) as being unpatentable over U.S. patent 5,960,202 (Granston et al.) and the Smatch documents. These rejections are improper and should be reversed.

Claim 11 is directed toward a method for using static analysis tools on a computer system to perform error analysis. With the method of claim 11, error analysis is performed on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process. The computer system has an operating system. The method of claim 11 involves running the build program on the computer system. The method of claim 11 also involves running a monitoring program on the computer system while the build program is running to compile the source code files. The monitoring program monitors activity between the build program and the operating system. The output from the monitoring program is used to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files.

The steps involved in using this technique to perform

static analysis are shown in FIG. 7 of applicants' patent application. As shown in FIG. 7, the monitoring program (also sometimes referred to as a trace program) is started at step 98. At step 100, the build program is run while using the trace program to monitor activity between the build program and the operating system. The information captured by the trace program is used to run a version of the build program with static analysis tools substituted for compilers (step 104).

In the Office Action, claim 11 was rejected under 35 U.S.C. §103(a) based on a combination of the Granston and Smatch references. In particular, it was suggested that Granston discloses applicants' claimed monitoring program and that Smatch teaches using the output from such a monitoring program to run a build program with static analysis tools substituted for compilers.

However, Granston does not disclose a monitoring program that produces an output suitable to run a build program as required by claim 11. Rather, Granston discloses a wrapper program that is stored in memory where the compiler usually resides (see column 3, lines 51-54). When the user runs a build program, the build program attempts to invoke the compiler (column 3, lines 59-61). Because the wrapper program is stored where the compiler program is normally stored, the wrapper program is invoked in place of the compiler (column 3, lines 61-

66). The wrapper program receives the compiler command line and determines which compiler options are contained in the command line (column 3, line 66 to column 4, line 1). The wrapper program then generates a log file that lists compiler options from the command line (column 4, lines 1-3). After generating the log file, the wrapper program invokes the compiler (column 4, lines 12-14).

In the Office Action, it was conceded that Granston fails to disclose using output from a monitoring program to run a build program with static analysis tools substituted for compilers so that static analysis tools perform static error analysis on the source code files. It was suggested that Smatch teaches this feature. In particular, the Office Action stated that Smatch teaches a system where a modified compiler and scripts are used to perform static analysis. According to the Office Action, it would be obvious to incorporate Smatch's modified compiler and scripts into Granston's system.

However, even if Smatch's modified compiler arrangement were to be used in conjunction with Granston's wrapper program as urged in the Office Action, the resulting combination would still fail to meet the limitations of claim 11. The proposed Granston/Smatch combination would have a wrapper program that logged compiler options while calling Smatch's xgcc compiler. But the Granston/Smatch combination

would not perform the claim 11 step of "using output from the monitoring program to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files." At no point would the output of Granston's wrapper program (the log file) be used to run Granston's build program. The only use of the log file output that is disclosed in Granston is to allow users to ascertain which compiler options are enabled in a build environment (see column 1, lines 46-54 of Granston). There is nothing in Smatch that makes up for this deficiency in Granston. In particular, nothing in Smatch shows or suggests using any output of a program to run a build program, let alone the output of a program such as Granston's wrapper program. Smatch requires users to hard code a new compiler name xgcc into their Makefile (see page 1 of SmUsing).

Applicants presented these arguments in their March 18, 2007 Reply. The Examiner responded on page 3 of the May 31, 2007 Office Action. In particular, the Examiner stated "It is noted that the output of Granston's wrapper is not just the log file as applicant asserts, but rather the wrapper also invokes the compiler using the compiler command line. Thus, Granston does teach using the output of the wrapper to run a build program, contrary to applicant's characterization. Accordingly, the teachings of Granston, when combined with the modified gcc

of Smatch, render claims 11-14 obvious." These arguments are unconvincing.

Because Granston's wrapper program is stored where the compiler usually resides, the wrapper program is invoked in place of the compiler when the build program attempts to invoke the compiler. The Granston wrapper program receives the compiler command line from the build program and determines which compiler options are contained in the command line. The wrapper program then generates a log file that lists compiler options from the command line. After generating the log file, the wrapper program invokes the compiler.

Granston does not teach the use of wrapper program output to run a build program as asserted by the Examiner. The statement by the Examiner that "Granston teaches using the output of the wrapper to run a build program" is simply not true.

As stated in Granston column 3, line 59 to column 4, line 3:

"When the user invokes the build process program (not shown), the build process program causes the CPU 1 to attempt to invoke the compiler program 11. However, since the wrapper program 20 is stored at the location in the memory device 2 where the compiler program 11 normally is stored, the CPU 1 causes the wrapper

program 20 to be invoked. The CPU 1 then passes the compiler command line to the wrapper program 20. The wrapper program analyzes the command line and determines which options are contained in the command line. The wrapper program 20 then generates a log file 30, which lists the compiler options contained in the command line."

As this passage in Granston makes clear, the build program invokes the wrapper program. The wrapper program does not invoke the build program.

Accordingly, even if Smatch's modified compiler arrangement were to be used in conjunction with Granston's wrapper program as urged in the Office Action, the resulting combination would still fail to meet the limitations of claim 11. The proposed Granston/Smatch combination would have a wrapper program that logged compiler options while calling Smatch's xgcc compiler. But the Granston/Smatch combination would not perform the claim 11 step of "using output from the monitoring program to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files." The wrapper program does not run a build program, let alone does the wrapper program run a build program with static

analysis tools substituted for compilers. Nothing in Smatch makes up for this deficiency in Granston. In particular, Smatch requires users to hard code a new compiler name xgcc into their Makefile, so Smatch does not show or suggest using any output of a program to run a build program, let alone the output of a program such as Granston's wrapper program.

Because the combination of Granston and Smatch does not meet all of the limitations of claim 11, claim 11 is patentable over Granston and Smatch, even if Granston and Smatch are combined as proposed in the Office Action. Claims 12-14 depend from claim 11 and are patentable because claim 11 is patentable. The rejections of claims 11-14 are therefore improper and should be reversed.

Argument for Independent Claim 15 and its Dependent Claims 16-18

Claims 15-18 were rejected under 35 U.S.C. §102(a) as being anticipated by Smatch. These rejections are improper and should be reversed.

Claim 15 is directed toward a method for using static analysis tools on a computer system to perform error analysis. The error analysis is performed on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process. The method of claim 15 involves redefining operating system commands in the

operating system. The build program is run on the computer system. As defined by claim 15, the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers so that the error analysis on the source code files is performed.

In the Office Action, it was suggested that the modified compiler xgcc that is used to compile code in Smatch performs the claimed method step of "redefining operating system commands in the operating system." (May 31, 2007 Office Action, page 8, citing SmIR at page 1.)

However, at no point in the SmIR document or any of the other Smatch documents is there any suggestion of redefining operating system commands in an operating system. The SmIR document describes how a modified compiler compiles source code and generates .c.sm files. A user then types a command `"/checker_script.pl file.c.sm"` to invoke a script. The script performs error analysis by analyzing the .c.sm files. At no point in the SmIR document or any of the other Smatch documents is there any suggestion to redefine operating system commands in an operating system.

Compilers are not operating systems. In particular, the xgcc compiler in Smatch is not an operating system nor is the xgcc compiler in Smatch part of an operating system. Because the compiler xgcc in Smatch is not an operating system

or part of an operating system, use of the compiler xgcc in Smatch does not teach applicants' claimed step of redefining operating system commands as urged in the Office Action.

Applicants' approach for redefining operating system commands is illustrated by step 108 of FIG. 8 in applicants' patent application and is described on pages 31 and 32 of applicants' patent application. As shown in step 108 of FIG. 8, operating system commands that may be redefined include process creation and execution commands such as fork and exec. Nothing in Smatch makes any mention of fork or exec or any other operating system commands, let alone does Smatch teach that one should redefine operating system commands as described by applicants in connection with FIG. 8 and as set forth in claim 15.

Applicants presented these arguments in their March 18, 2007 Reply. In the May 31, 2007 Office Action, the Examiner responded by agreeing that compilers are not operating systems. Nevertheless, the Examiner asserted "that compilers that may be invoked through operating system command lines may reasonably be considered operating system commands as recited in claims 15-18." Thus, even though the Examiner has conceded that compilers are not operating systems, the Examiner has asserted that compilers are operating system commands.

The Examiner's reasoning is unconvincing. As

applicants have already explained, compilers are not operating systems, so operating system commands have nothing to do with compilers. Moreover, the language of claim 15 clearly requires the step of "redefining operating system commands in the operating system." The Examiner has glossed over the limitations of claim 15 by ignoring the language "in the operating system." The claim term at issue is not "operating system commands," but rather "operating system commands in the operating system." The Examiner has not explained (nor even attempted to explain) how a compiler could possibly be considered to be "an operating system command in the operating system." Claim 15 is therefore allowable over Smatch.

Claim 15 is also allowable over Smatch, even if Smatch's xgcc modified compiler were to be considered an "operating system command" as urged by the Examiner. Claim 15 recites a method step of "running the build program on the computer system, wherein the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers so that the error analysis on the source code files is performed." Although Snatch discloses creating a modified compiler xgcc, Snatch does not disclose that the naming of a modified compiler xgcc (instead of gcc) results in the redefined operating system commands causing a build program to invoke the static analysis tools. The xgcc program in Snatch is

invoked by manually changing the variable cc. The renaming of gcc to xgcc does not cause a build program to invoke xgcc in place of gcc.

Moreover, the language of claim 15 is plural (e.g., static analysis tools, compilers), whereas Smatch at most discloses a single compiler (gcc) and a single alleged static analysis tool (xgcc).

On page 4, lines 1-6 of the May 31, 2007 Office Action, the Examiner suggested that applicants' March 18, 2007 Reply contained arguments that were inappropriately directed to limitations that were not present in the claims. In particular, the Examiner stated that the operating system fork and exec commands that were discussed by applicants were not to be read into the claims as limitations. However, applicants never suggested that the claimed method step of "redefining operating system commands in the operating system" applied only to the fork and exec commands. Those operating system commands were described for the Examiner as an example. Although applicants pointed out that nothing in Smatch makes any mention of fork or exec, applicants also pointed out that Smatch makes no mention of any other operating system commands.

Because Smatch fails to disclose applicants' claimed method step of redefining operating system commands in an operating system, claim 15 is not anticipated by Smatch. Claim

15 is therefore in condition for allowance. Claims 16-18 depend from claim 15 and are allowable because claim 15 is allowable. The Examiner's rejections of claims 16-18 are in error and should be reversed.

Conclusion

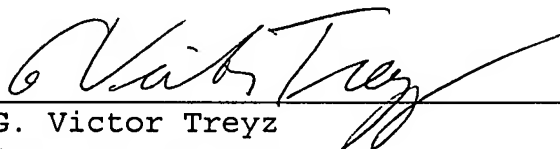
The foregoing demonstrates that applicants' claims are patentable over the Smatch and Granston references.

Claims 1-18 are in therefore in condition for allowance. The Examiner's rejections of these claims should be reversed.

Respectfully submitted,

7/13/07

Date



G. Victor Treyz

Reg. No. 36,294

Attorney for Applicants

Customer No. 36532

CLAIMS APPENDIX

1. A method for using static analysis tools on a computer system to perform error analysis on source code files, comprising:

running a build program on the computer system to invoke compilers on the computer system that compile the source code files into executable code, wherein running the build program produces a build program output; and

running a static analysis tool management program on the computer system to invoke the static analysis tools and produce corresponding static error analysis results, wherein the static analysis tool management program accepts the source code and the build program output as inputs.

2. The method defined in claim 1 further comprising directing the build program output to a file that is used as an input by the static analysis tool management program.

3. The method defined in claim 1 further comprising directing the build program output to a file that is used as an input by the static analysis tool management program, wherein the file contains information on which static analysis tools to substitute for each compiler when the static analysis tools are invoked.

4. The method defined in claim 1 further comprising providing a user with an opportunity to specify for the static analysis tool management program which compiler options should be ignored by the static analysis tools when performing static analysis on the source code.

5. The method defined in claim 1 further comprising providing a user with an opportunity to specify for the static analysis tool management program which additional compiler options are required by the static analysis tools when performing static analysis on the source code.

6. The method defined in claim 1 further comprising invoking a plurality of build management utilities with the build program as the build program is run, wherein the build program output includes output from the build management utilities.

7. A method for using static analysis tools on a computer system to perform error analysis on source code files that may be compiled on the computer system using compilers having corresponding compiler names that are invoked using a build program during a build process, comprising:

creating a new directory on the computer system;
modifying a search path on the computer system so
that the new directory is included first in the search path;
placing the static analysis tools into the new
directory, wherein the static analysis tools in the new
directory are given names matching the compiler names; and
running the build program so that the static
analysis tools with the names matching the compiler names are
invoked.

8. The method defined in claim 7 comprising:
obtaining information from a user on compilation
options for the compilers; and
using the information on the compilation options
when invoking the static analysis tools by running the build
program.

9. The method defined in claim 7 wherein running the
build program comprises making calls to the compiler names.

10. The method defined in claim 7 wherein running the
build program invokes both the compilers and the static analysis
tools.

11. A method for using static analysis tools on a computer system to perform error analysis on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process, wherein the computer system has an operating system, the method comprising:

running the build program on the computer system;
running a monitoring program on the computer system while the build program is running to compile the source code files, wherein the monitoring program monitors activity between the build program and the operating system; and

using output from the monitoring program to run the build program with the static analysis tools substituted for the compilers so that the static analysis tools perform static error analysis on the source code files.

12. The method defined in claim 11 further comprising gathering information from a user as to which compilers are used during the build process and which compilation options are to be used by the static analysis tools.

13. The method defined in claim 11 further comprising filtering the output from the monitoring program to remove compiler option commands.

14. The method defined in claim 11 wherein running the monitoring program comprises running a custom monitoring program that uses operating system debugging commands to monitor the activity between the build program and the operating system.

15. A method for using static analysis tools on a computer system to perform error analysis on source code files that may be compiled on the computer system using compilers that are invoked using a build program during a build process, wherein the computer system has an operating system comprising:

redefining operating system commands in the operating system; and

running the build program on the computer system, wherein the redefined operating system commands cause the build program to invoke the static analysis tools in place of the compilers so that the error analysis on the source code files is performed.

16. The method defined in claim 15 further comprising using user-specified information on the compilers and compiler options during invocation of the static analysis tools.

17. The method defined in claim 15 wherein redefining the operating system commands comprises redefining operating system process creation and execution commands by placing modified versions of the operating system creation and execution commands on the computer system and by instructing the operating system to load the modified versions of the operating system process creation and execution commands.

18. The method defined in claim 15 wherein redefining the operating system commands comprises using a new kernel module containing modified functions.

EVIDENCE APPENDIX

None

RELATED PROCEEDINGS APPENDIX

None